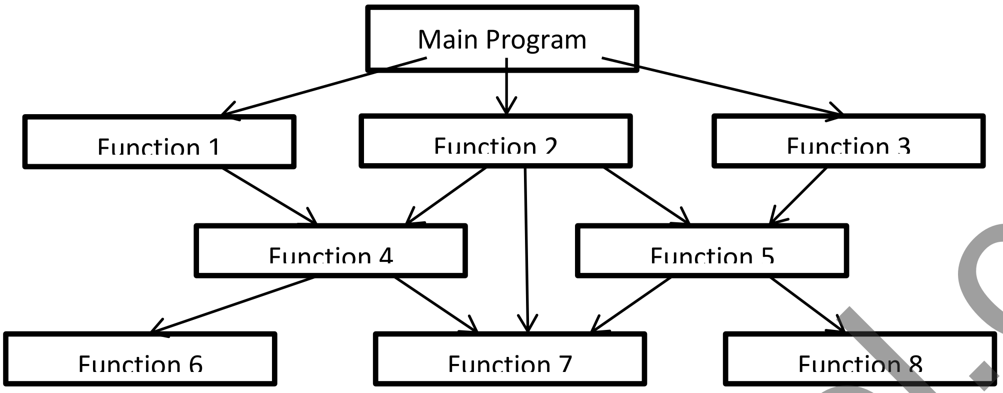
Unit1: Concepts of OOPS:

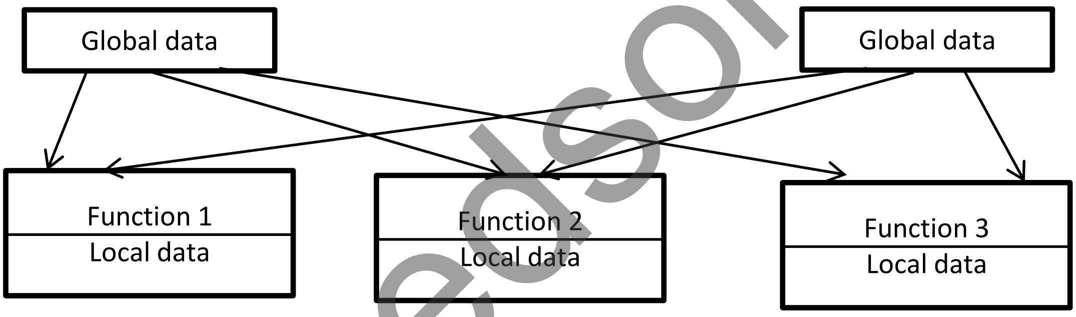
Unit1.1: Difference between procedural programming and OOPS

POP (Procedural Oriented Programming Language)

A word Procedure means a set of sub – routine (Instructions) or a set of function. Function means set of instruction to perform particular task. In POP we give more focus on development of function rather than data.



In this multi-functional program, many important data are placed as a global data so that they may be accessed by all the functions. This type of function may have their own local data also.



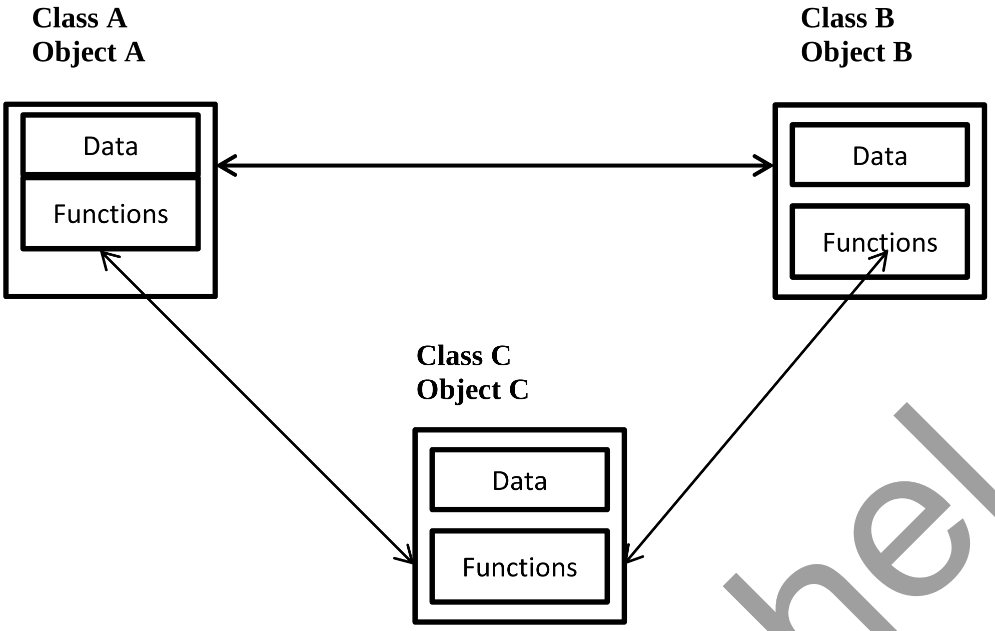
Characteristics:

- Puts much importance on things to be done.
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transfer data from one form to another.
- Employs top down approach in program designing.
- In the cases of large program, bringing change is difficult and time consuming.
- Appropriate and effective techniques are unavailable to secure data of a function from others.

Drawback:
In multi-function program more than one function can access global data. Here, the problem is if one function accidentally changes the global data then, it will affect all the other functions which are using that global data. It is also difficult to develop and to identify that which function is creating the problem.

OOP (Object Oriented Programming Language)

OOP is a language which provides a way to modularise the program by creating partition of memory for both data and function. OOP gives more importance to data and does not allow them to flow freely around the system. It ties the data more closely to the function and protects them from accidental changes. OOP decompose the problem unto number of entities which is called objects. The data of the object can be accessed by the function which is associated with same type of object. However, function of one object can access function of another object.



Difference between POP and OOP

Features	POP	OOP
Division	Program is divided into small parts called function.	Program is divided into small part called object.
Importance	Importance is given to functions rather than data.	Importance is given to data rather than functions.
Approach	It follows top to bottom approach.	It follows bottom to top approach.
Specifies	POP does not have any specifiers.	OOP has access specifiers like public, private, protected etc.
Data movement	It allows free movement of data from function to function.	Data cannot move freely into the system. (it is closely attach with the function)
Expansion	It is not easy to add new data or functions in the program.	It provides easy way to add new data or function.
Data hiding	It does not provide a way to hide the data.	It provides a way to hide the data so that it is more secure.
Overloading	Overloading is not possible.	Overloading is possible in the form of function overloading and operator overloading.
Example	C, COBOL, Fortran	C++, Java, .NET

Features:

- Object
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

Benefits of OOP

- Through Inheritance we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working programs that communicate with rather than having to start writing the code from the scratch.
- This will help in saving development time and cost.
- The principal of Data hiding helps the programmer to build a secure program that cannot be changed by code in other parts of program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- OOP systems can be easily upgraded from small to large.
- Software complexity can be easily managed.

Unit 1.2: Various library(header) files require for C++

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions to objects like cin, cout, cerr etc.
2. **iomanip**: iomanip stands for input output manipulators. The methods declared in these files are used for manipulating streams. This file contains definitions of setw, setprecision, etc.
3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.
4. **String.h:** It is used to perform various functionalities related to string manupulation like strlen(), strcmp(), strcpy(), size(), etc.
5. **math:** It is used to perform mathematical operations like sqrt(), log2(), pow(), etc.

Iostream.h

In C++ input and output are performed in the form of a sequence of bytes or more commonly known as **streams**.

- **Input Stream:** If the direction of flow of bytes is from the device (for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device (display screen) then this process is called output.

The two keywords **cout in C++** and **cin in C++** are used very often for printing outputs and taking inputs respectively. These two are the most basic methods of taking input and printing output in C++. To use cin and cout in C++ one must include the header file *iostream* in the program.

Standard output stream (cout): Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator (<<).

standard input stream (cin): Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **iostream** and is used to read input from the standard input device which is usually a keyboard. The extraction operator (>>) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keyboard.

Main() function of C++

```
#include<iostream.h>
#include<conio.h>

int main()
{
        clsrscr();
        cout<< ”Sample Program”;
        getch();
        return 0;
}
```

Input and Output operators:

Syntax:

```
cout << " Sample Program";
```

The above statement will display the string which is in quotation mark. This statement introduces new C++ feature "cout" and "<<". The identifier cout is predefined object that represent the standard output string. The operator "<<" is called "insertion" or "put to" operator.

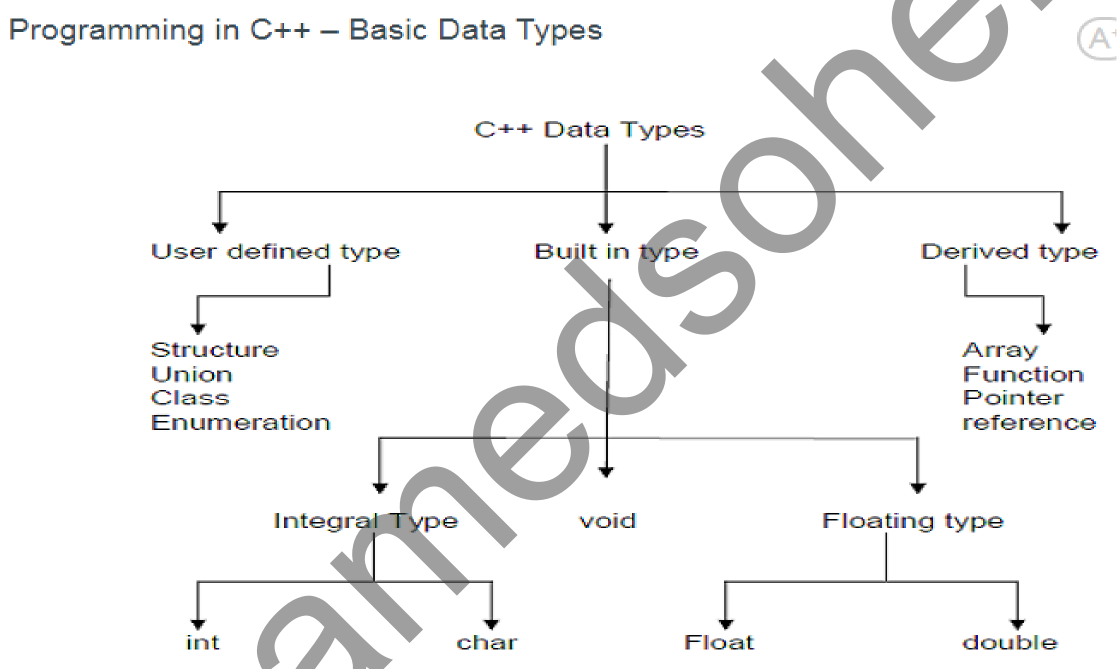
```
cin >>number;
```

It is an input statement which is used to accept the value from standard input. The identifier "cin" is predefined object that represent the standard input stream. The operator ">>" known as "extraction" operator or "get from" operator. It extracts the value from keyboard and assigns it to the variable.

We can also cascade input and output.

```
cout<<"sum = "<<sum<<"rs";
cin>>no1>>no2;
```

Unit 1.3 Data types in C++:



Built-in-type

1) Integral type : – The data types in this type are int and char. The modifiers signed, unsigned, long & short may be applied to character & integer basic data type. The size of int is 2 bytes and char is 1 byte.

2) void – void is used :

i) To specify the return type of a function when it is not returning any value.

ii) To indicate an empty argument list to a function.

Ex- void function1(void)

iii) In the declaration of generic pointers.

Ex- void *gp

A generic pointer can be assigned a pointer value of any basic data type.

Ex . int *ip // this is int pointer

gp = ip //assign int pointer to void.

A void pointer cannot be directly assigned to their type pointers in c++ we need to use cast operator.

Ex – void *ptr1;

char *ptr2;

ptr2 = ptr1; is allowed in c but not in c++.

ptr2 = (char *)ptr1; is the correct statement.

3) Floating type:

The data types in this are float & double. The size of the float is 4 byte and double is 8 byte. The modifier long can be applied to double & the size of long double is 10 byte.

User-defined type:

i) The user-defined data type structure and union are same as that of C.

ii) **Classes** – Class is a user defined data type which can be used just like any other basic data type once declared. The class variables are known as objects.

iii) **Enumeration**

a) An enumerated data type is another user defined type which provides a way of attaching names to numbers to increase simplicity of the code.

b) It uses enum keyword which automatically enumerates a list of words by assigning them values 0, 1, 2,…..etc.

Syntax:-

```
        enum shape {
circle, square, triangle;
}
```

Now shape becomes a new type name & we can declare new variables of this type.

Ex . shape oval;

d) In C++, enumerated data type has its own separate type. Therefore c++ does not permit an int value to be automatically converted to an enum value.

```
Ex. shape shapes1 = triangle, // is allowed
shape shape1 = 2; // Error in c++
shape shape1 = (shape)2; //ok
```

e) By default, enumerators are assigned integer values starting with 0, but we can override the default value by assigning some other value.

Ex enum colour {red, blue, pink = 3}; //it will assign red to o, blue to 1, & pink to 3 or enum colour {red = 5, blue, green}; //it will assign red to 5, blue to 6 & green to 7.

Derived Data types:

1) Arrays

An array in c++ is similar to that in c, the only difference is the way character arrays are initialized. In c++, the size should be one larger than the number of character in the string where in c, it is exact same as the length of string constant.

```
Ex – char string1[3] = "ab"; // in c++
char string1[2] = "ab"; // in c.
```

2) Functions

Functions in c++ are different than in c there is lots of modification in functions in c++ due to object orientated concepts in c++.

3) Pointers

Pointers are declared & initialized as in c.

Ex int * ip; // int pointer
ip = &x; //address of x through indirection

c++ adds the concept of constant pointer & pointer to a constant pointer.
char const *p2 = .HELLO.; // constant pointer

Next you have a summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

Name Description Size* Range*
char – 1byte; signed: -128 t127, unsigned: 0 to 255
short int (short)
short Integer – 2bytes; signed: -32768 to 32767, unsigned: 0 to65535

int Integer. 4bytes signed: – 2147483648 to 2147483647, unsigned: 0 to 4294967295

long int or (long)

Long integer. 4bytes signed: -2147483648 to 2147483647, unsigned: 0 to 4294967295

Bool – Boolean value. It can take one of two values: true or false.
1byte true or false
Float Floating point **number**.- 4bytes 3.4e +/- 38 (7 digits)

double Double precision
floating point number.

Unit 1.4 Concepts of String:

Unit 1.4.1 Character Array:

In C programming, the collection of characters is stored in the form of **arrays**. Hence, it's called C-strings. C-strings are **arrays** of type char terminated with null **character**, that is, \0.
Declaration
Syntax: char variable_name[n];

char ch[5];

The above statement will create a character array ch of length 5.
We can also declare and define character array together.
Char myword[] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
The above declares an **array** of 6 elements of type **char** initialized with the **characters** that form the word "Hello" plus a null **character** '\0' at the end.

Example:

```
#include<iostream.h>
#include<conio.h>
int main()
{
        char firstName[30], secondName[30];

        //input values
        cout<<"What is your first name? ";
        cin>>firstName;
        cout<<"What is your second name? ";
        cin>>secondName;

        //printing
        cout<<"Hi "<<firstName<<" "<<secondName<<endl;

        return 0;
}
```

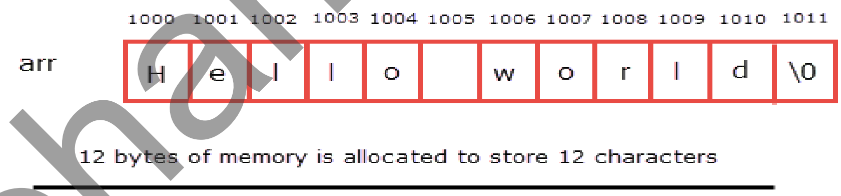
<u>Unit 1.4.2 pointer to character array:</u>

```
char arr[] = "Hello World"; // array version
char ptr* = "Hello World";  // pointer version
```

The type of both the variables is a pointer to char or (char*), so you can pass either of them to a function whose formal argument accepts an array of characters or a character pointer.
Here are the differences:

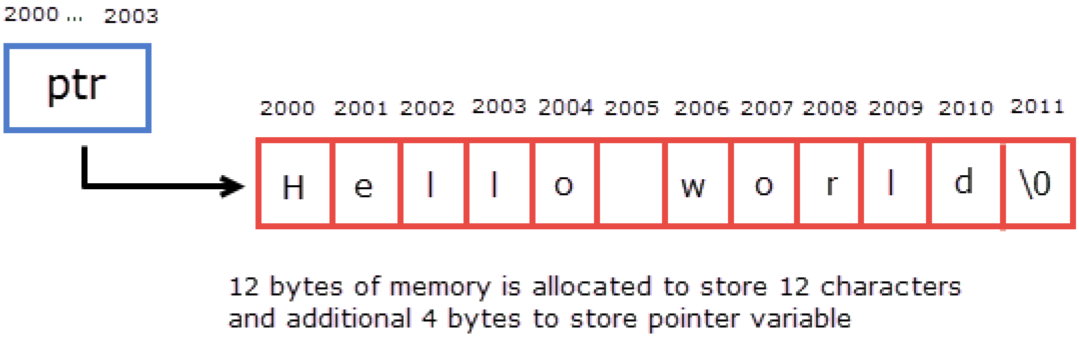
1. arr is an array of 12 characters. When compiler sees the statement:
2. char arr[] = "Hello World";

It allocates 12 consecutive bytes of memory and associates the address of the first allocated byte with arr.



12 bytes of memory is allocated to store 12 characters

On the other hand when the compiler sees the statement.
char ptr* = "Hello World";
It allocates 12 consecutive bytes for string literal "Hello World" and 4 extra bytes for pointer variable ptr. And assigns the address of the string literal to ptr. So, in this case, a total of 16 bytes are allocated.



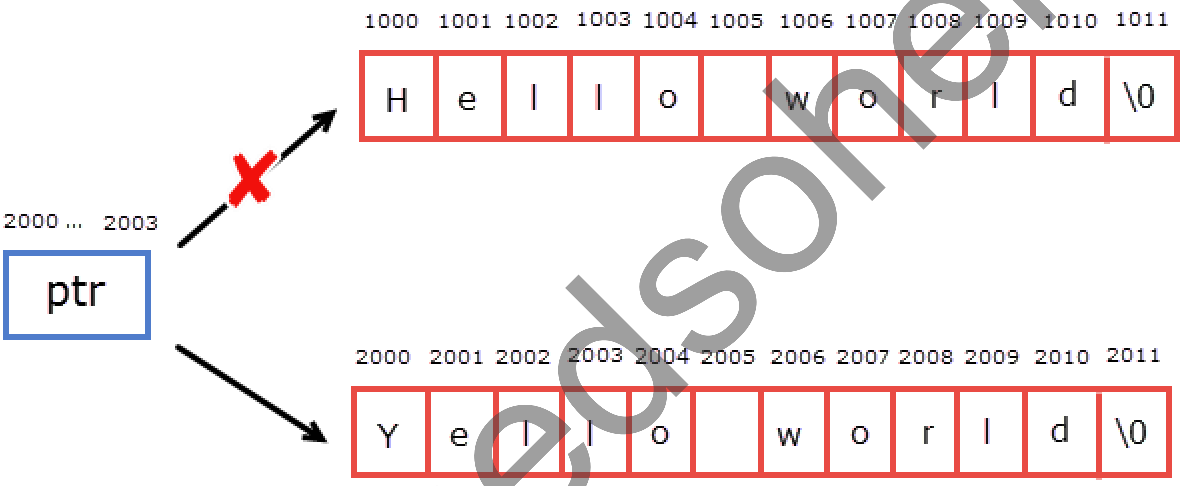
12 bytes of memory is allocated to store 12 characters
and additional 4 bytes to store pointer variable

We already learned that name of the array is a constant pointer. So if arr points to the address 1000, until the program ends it will always point to the address 1000, we can't change its address. This means string assignment is not valid for strings defined as arrays.

arr = "yellow World"; *// Wrong*

On the contrary, ptr is a pointer variable of type char, so it can take any other address. As a result string, assignments are valid for pointers.

ptr = "Yellow World"; *// ok*



After the above assignment, ptr points to the address of "Yellow World" which is stored somewhere in the memory.

Unit 1.4.3 Use of String.h and its important functions:

The **string.h** header defines various functions for manipulating arrays of characters. Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

Some of the most commonly used String functions are:

strcat:

The strcat() function will append a copy of the source string to the end of destination string. The strcat() function takes two arguments:

1) dest
2) src

It will append copy of the source string in the destination string. **The terminating character at the end of dest is replaced by the first character of src .**

Return value: The strcat() function returns dest, the pointer to the destination string.

```
CPP program to demonstrate strcat

#include <string.h>
#include <iostream,h>
int main()
{
   char dest[50] = "This is an";
   char src[50] = " example";

   strcat(dest, src);
   cout << dest;
   return 0;
}
```

Output:

```
This is an example
```

strcmp:

strcmp() is a built-in library function and is declared in **<string.h>** header file. This function takes two strings as arguments and compare these two strings lexicographically.

Syntax::

int strcmp(const char *leftStr, const char *rightStr);

In the above prototype, function srtcmp takes two strings as parameters and returns an integer value based on the comparison of strings.

- strcmp() compares the **two strings lexicographically** means it starts comparison character by character starting from the first character until the characters in both strings are equal or a NULL character is encountered.
- If first character in both strings are equal, then this function will check the second character, if this is also equal then it will check the third and so on
- This process will be continued until a character in either string is NULL or the characters are unequal.

```
// C program to illustrate strcmp() function


  #include <stdio.h>
  #include <string.h>

  int main()
  {

      char leftStr[] = "g f g";
      char rightStr[] = "g f g";

      // Using strcmp()
      int res = strcmp(leftStr, rightStr);

      if (res == 0)
         cout<<"Strings are equal"
      else
         cout<<"Strings are unequal”

      cout<<"Value returned by strcmp() is:<<res
      return 0;
  }
```

Output:

```
Strings are equal
Value returned by strcmp() is:  0
```

strcpy:

strcpy() is a standard library function in C/C++ and is used to copy one string to another. In C it is present in **string.h** header file and in C++ it is present in **cstring** header file.

Syntax:
char* strcpy(char* dest, const char* src);
Paramters: This method accepts following paramters:

- **dest**: Pointer to the destination array where the content is to be copied.
- **src:** string which will be copied.

Return Value: After copying the source string to the destination string, the strcpy() function returns a pointer to the destination string.
Below program explains different usages of this library function:

```
// C program to illustrate strcpy() function ic C/C++

#include <iostream.h>
#include <string.h>

int main()
{
   char str1[] = "Hello!";
   char str2[] = "Students";
   char str3[40];
   char str4[40];
   char str5[] = "GfG";

   strcpy(str2, str1);
   strcpy(str3, "Copy successful");
   strcpy(str4, str5);
   cout<<"str1: “<< str1<<"str2: “<< str2<<"str1: “<< str3<<"str4: “<<str4
   return 0;
}
```

Output:

```
str1: Hello!
str2: Students
str3: Copy successful
str4: GfG
```

strlen:

The **strlen()** function calculates the length of a given string.The **strlen()** function is defined in **string.h** header file. It doesn’t count null character ‘\0’.

Syntax:
int strlen(const char *str);
Parameter:

- **str:** It represents the string variable whose length we have to find.

Return: This function returns the length of string passed.
Below programs illustrate the strlen() function in C:

```
// c program to demonstrate example of strlen() function.

#include <iostram.h>
#include <string.h>

int main()
{
    char ch[] = { 'g', 'e', 'e',
                  'k', 's', '\0' };

strlen(ch));
Cout<<”Length of string is: "<<ch;


    return 0;
}
```

- **Output:**
- Length of string is: 5

strncat:

In C/C++, strncat() is a predefined function used for string handling. **string.h** is the header file required for string functions.

This function appends not more than **n** characters from the string pointed to by src to the end of the string pointed to by dest plus a terminating Null-character. The initial character of string(src) overwrites the Null-character present at the end of string(dest). Thus, length of the string(dest) becomes strlen(dest)+n. But, if the length of the string(src) is less than **n**, only the content up to the terminating null-character is copied and length of the string(dest) becomes strlen(src) + strlen(dest).

The behavior is undefined if

- the strings overlap.
- the dest array is not large enough to append the contents of src.

Syntax:

char *strncat(char *dest, const char *src, size_t n)

Parameters: This method accepts following paramters:

- **dest**: the string where we want to append.
- **src**: the string from which ‘n’ characters are going to append.
- **n**: represents maximum number of character to be appended. size_t is an unsigned integral type.

Return Value: The strncat() function shall return the pointer to the string(dest).

Program:

```
// C, C++ program demonstrate functionality of strncat()

#include <iostream.h>
#include <string.h>

int main()
{

   // Take any two strings
   char src[50] = "efghijkl";
   char dest[50] = "abcd";

   // Appends 5 character from src to dest
   strncat(dest, src, 5);
```

```
    // Prints the string
    Cout<<"Source string :”<< src;
    Cout<<"Destination string : “<< dest;

    return 0;
}
```

Output:

```
Source string : efghijkl
Destination string : abcdefghi
```

strncmp:

This function lexicographically compares not more than count characters from the two null-terminated strings and returns an integer based on the outcome.

- This function takes two strings and a number **num** as arguments and compare at most first **num** bytes of both the strings.
- **num** should be at most equal to the length of the longest string. If **num** is defined greater than the string length than comparison is done till the null-character(‘\0’) of either string.
- This function compares the two strings lexicographically. It starts comparison from the first character of each string. If they are equal to each other, it continues and compare the next character of each string and so on.
- This process of comparison stops until a terminating null-character of either string is reached or **num** characters of both the strings matches.

Syntax :

int strncmp(const char *str1, const char *str2, size_t count);

Parameters:

str1 and str2: C string to be compared.
count: Maximum number of characters to compare.
size_t is an unsigned integral type.

Return Value:

Value	**Meaning**
Less than zero	str1 is less than str2.
Zero	str1 is equal to str2.
Greater than zero	str1 is greater than str2.

Example:

```
// C, C++ program to demonstrate
// functionality of strncmp()

#include <iostream.h>
#include <string.h>

int main()
{
    // Take any two strings
    char str1[10] = "aksh";
    char str2[10] = "akash";

    // Compare strings using strncmp()
    int result = strncmp(str1, str2, 4);

    if (result == 0) {
        // num is the 3rd parameter
        // of strncmp() function
        Cout<<"str1 is equal to str2 upto num characters";
    }
    else if (result > 0)
        cout<<"str1 is greater than str2";
    else
        cout<<"str2 is greater than str1";

    cout<<"Value returned by strncmp() is: "<<result;

    return 0;
}
```

Output:

```
str1 is greater than str2
Value returned by strncmp() is: 18
```

strncpy:

The strncpy() function is similar to strcpy() function, except that at most n bytes of src are copied. If there is no NULL character among the first n character of src, the string placed in dest will not be NULL-terminated. If the length of src is less than n, strncpy() writes additional NULL character to dest to ensure that a total of n character are written.

Syntax:

char *strncpy(char *dest, const char *src, size n)

Parameters: This function accepts two parameters as mentioned above and described below:

- **src:** The string which will be copied.
- **dest:** Pointer to the destination array where the content is to be copied.
- **n:** The first n character copied from src to dest.

Return Value: It returns a pointer to the destination string.

Example:

```
// C Program  to illustrate the strcpy() function in C/C++
#include <iostream.h>
#include <string.h>
int main()
{
    char src[] = "hello world";

    // The destination string size is 14.
    char dest[14];

    // copying n bytes of src into dest.
    strncpy(dest, src, 5);
    cout<<"Copied string:"<< dest;

    return 0;
}
```

Output:

```
Copied string: hello
```

1.5 Concepts of Class and Objects

Class:

The entire set of data and function of an object create a user defined data type with the help of class. Class is a user defined data type which binds data and function together. Class is a collection of similar type of object and objects are variables of class. Once the class is defined you can create as many objects as you want. A syntax of creation of object is similar to creation of any variable. When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows –

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

Object:

Objects are basic runtime entity, they may represent a person, place, bank account, table of data or any item which can be handled by program. Objects take space in memory and have address associated with it. They also represent user defined data such as array, vectors and lists. Each object contains code and data. An object can interact with another object without knowing the details of each other. It is sufficient to know type of message and the type of response returned by those objects. A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box −

```
Box Box1;          // Declare Box1 of type Box
Box Box2;          // Declare Box2 of type Box
```